Preface or Abstract

Project & IPDT Discussion

The AeroSpec Engine division is divided into to major work packs, namely the Compressor Design Team and the Turbine Design Team. The team split is as follows:

Contents

List of Figures

List of Tables

Nomenclature

To be added with Moustoupha type.

Part 1

High Pressure Compressor Design

Part 2

High Pressure Turbine Design

This section covers the aerodynamic design and performance analysis of the high pressure axial turbine. The methodology discussed in the following subsections is presented in the chronological order of the design.

2.1 Meanline Design

RAFAL, PARAM and ZAC

2.2 Structural Design

RAFAL

2.3 Blade Vortex Design

ZAC

2.4 Aerodynamic Losses

Turbine design optimization study must account for flow losses. Losses can be calculated using viscous, three-dimensional CFD analysis, but this approach has drawbacks due to its place in the design cycle [1]. Until precise blade shapes are known at a later stage, CFD analysis is not possible. Hence, a meanline analysis incorporating aerodynamic losses as functions of blade row inlet and exit velocity triangles, and overall geometric features is used for the scope of this analysis. The methodology is adapted from reference [1].

The profile loss is the result of skin friction on the surface and depends on the blade's contact area with the fluid, surface finish, Reynolds number, and Mach number of the flow. These effects are influenced by the airfoil's geometry. Annulus losses are also caused by friction on the endwall surfaces [1].

Secondary flow, which consists of vortices resulting from boundary layers and passage curvature, causes some fluid to move in directions other than the main flow direction. Trailing vortices are formed

at the blade's trailing edge when the flow separates, creating a wake. The wake's momentum deficit and the kinetic energy in the trailing vortices contribute to losses. Tip clearance losses occur in rotors when fluid leaks into the gap between the blade tip and the shroud, contributing little to no expansion work. The resulting tip leakage vortex interacts with corner and passage vortices, creating complex flow patterns within the passage.

Hence, the total pressure loss of a blade row is calculated as the sum of different losses as mentioned previously in this section. Equation 2.1 is the relation used.

$$K_T = K_p f_{Re} + K_s + K_{TE} + K_{clr} \tag{2.1}$$

2.4.1 Profile Loss

Profile loss coefficient is calculated based on a set of cascade set results. The profile loss coefficient is calculated as follows:

$$K_p = 0.914 \left(\frac{2}{3} K_p^{\star} K_{accel} + K_{sh} \right) \tag{2.2}$$

$$K_p^{\star} = \left\{ K_{P(\beta_1=0)} \left| \frac{\beta_1}{\alpha_2} \right| \left(\frac{\beta_1}{\alpha_2} \right) \left[K_{P(\beta_1=\alpha_2)} - K_{P(\beta_1=0)} \right] \right\} \left(\frac{t_{max}/c}{0.2} \right)^{(\beta_1/\alpha_2)} \tag{2.3}$$

Equation 2.3 can be used to interpolate for any given angle β_1 and α_2 using the graphs provided in reference [1]. To make the design process faster and to calculate the values for multiple design choices, the graphs were digitized and added to a database, where a linear regression machine learning model is used to provide the value for any given input.

Figure 2.1 presents a sample code for predicting the loss coefficients based on the provided. Since the loss calculation model discussed in Chapter 2 of reference [1] uses a lot of correlations which have been presented in forms of graphs, the same technique is used to predict the required values and will not be discussed again in the following sections.

```
    def figure_2_3a(pitch_chord_ratio, exit_flow_angle):
        fig_2_3a = pd.read_csv(r'_input_database\figure_2_3a.csv')
        X = fig_2_3a[['pitch_chord_ratio', 'exit_flow_angle']]
        y = fig_2_3a['K_P_1']

        X, X_test, y, y_test = train_test_split(X, y, test_size=0.2)

        model = LinearRegression()
        model.fit(X_train, y_train)
        K_P_1 = model.predict(X)

        return K_P_1[0]
```

Figure 2.1: Example code for calculating profile loss coefficient, $K_{P,1}$

$$K_{sh} = \left(\frac{\Delta p_0}{q_1}\right)_{sh} \left(\frac{p_1}{p_2}\right) \frac{1 - \left(1 + \frac{\gamma-1}{2} M_1^2\right)^{\gamma/(\gamma-1)}}{1 - \left(1 + \frac{\gamma-1}{2} M_2^2\right)^{\gamma/(\gamma-1)}} \tag{2.4}$$

where,

$$\left(\frac{\Delta p_0}{q_1}\right)_h = 0.75(M_{1,hub} - 0.4)^{1.75}$$

and

$$\left(\frac{\Delta p_0}{q_1}\right)_{sh} = \left(\frac{r_h}{r_t}\right) \left(\frac{\Delta p_0}{q_1}\right)_h$$

The effect of exit Mach number and channel flow acceleration is corrected for using the following relations:

$$\begin{aligned} K_{accel} &= 1 - K_2(1 - K_1) \\ K_1 &= \begin{cases} 1.0 & \text{for } M_2 \leq 0.2 \\ 1 - 1.25\,(M_2 - 0.2) & \text{for } M_2 > 0.2 \end{cases} \\ K_2 &= (M_1/M_2)^2 \end{aligned} \tag{2.5}$$

2.4.2 Secondary Loss

2.4.3 Trailing Edge Loss

2.4.4 Tip Clearance Loss

2.4.5 Efficiency Calculation

$$\eta_{tt} = \frac{1}{\left\{1 + \left[\frac{\zeta_N V_2^2 + \zeta_R V_{r3}^2}{2(h_{01} - h_{03})}\right]\right\}} \tag{2.6}$$

$$\zeta_R = \frac{K_T}{1 + 0.5\gamma M_{r3}^2} \tag{2.7}$$

Part 3

Converged Design

Part 4

Trade Studies

Part 5

Conclusion

Bibliography

[1] Hany Moustapha. *Axial and Radial Turbines*. Concepts NREC, White River Junction, Vt., 2003.

APPENDICES

Appendix A

Python Script

Space left blank on purpose.

In [ ]:
```
import math
import numpy as np
```

Define Engine Characteristics

In [ ]:
```
# Provided Engine Characteristics
mechanical_eff = 0.99
gamma = 1.4
gamma_g = 1.33333
c_p_air = 1.005
c_p_gas = 1.148
# ENGINE INLET
inlet_loss = 0.01           # Inlet pressure loss
# ------------------------------------------------------------------------------
# Compressor
m_dot = 5.21                # [lb/s]
lpc_pr = 4                  # LPC PR
lpc_eff = 0.865             # LPC Target Efficiency
hpc_pr = 3                  # HPC PR
hpc_eff = 0.855             # HPC Target Efficiency
hpc_bleed_air = 0.09        # HPC Bleed Air (exit)
# ------------------------------------------------------------------------------
# Combustor
AFR = 0.02                  # Fuel to Air Ratio or Air Fuel Ratio (AFR)
FHV = 40007.2               # [kJ/kg] Fuel Heating Value (FHV)
combustor_eff = 0.99
combustor_loss = 0.018      # Combustor pressure loss
RTDF = 0.05                 # Radial Temperature Distribution Factor (RTDF)
# ------------------------------------------------------------------------------
# Turbine
hpt_eff = 0.83              # HPT Target Efficiency->Given range 0.83-0.85
hpt_vane_cooling = 0.03     # HPT Vane Cooling Air
hpt_disk_cooling = 0.165    # HPT Disk Cooling Air
lpt_eff = 0.91              # LPT Target Efficiency
hpt_vane_cooling = 0.011    # LPT Vane Cooling Air
ITD_loss = 0.006            # ITD? Loss
pt_eff = 0.92               # PT Target Efficiency
pt_disk_cooling = 0.0125    # PT disk cooling air
# ------------------------------------------------------------------------------
# Exhaust
exhaust_loss = 0.02         # Exhaust Loss
exhaust_mach = 0.15         # Exhaust Mach Number
```

Functions for calculations

In [ ]:
```
def calc_turbine_pressure(P_i, T_03, T_04, eta_turbine):
    """
    This function calculation the exit pressure of a turbine using the isentropic efficiency.
    Inputs: P_i, T_03, T_04, eta_turbine
    Outputs: P_f

    """
    temp_val_1 = 1.33333/(1.33333 - 1)
    temp_val_2 = 1 - ((T_03 - T_04) / (eta_turbine * T_03))
    P_f = P_i * (temp_val_2**temp_val_1)
    print(P_f)
    return P_f

def calc_psi(c_p, delta_T_0_s, U):
    """
    This function calculates the blade loading coefficient

    ____________
    Input: c_p, delta_T_0_s, U
    """
    psi = (2 * c_p * delta_T_0_s)/(U**2)
    print("psi = ", psi)
    return psi

def calc_lambda(alpha, phi, psi):
    """
    This function calculates the degree of reaction

    ____________
    Input: Swirl angle "alpha", flow coefficient "phi", blade loading coefficient "psi"

    ____________
    Output: Returns the value of the degree of coefficient and the tan_beta_3
    """
    alpha = math.radians(alpha)
    tan_beta_3 = math.tan(alpha) + (1/phi)
    lambda_val = (4 * tan_beta_3 * phi - psi)/4
    print("lambda = ", lambda_val)
    return lambda_val

def calc_T_static(T_total, M):
    T_static = T_total/(1 + 0.15*(M**2))
```

```
    return T_static


def calc_area(T_static, p_static, C_a, n):
    """
    This function calculate air properties and also the area from mass flow rate
    ____________
    Input: Total Temperature "T_01", Total Pressure "p_01", Flow velocity "c", axial component of the velocity "c_a", station
    ____________
    Output: A plot showing stability in real and imaginary axis.
    """
    rho_ = p_static/(R*T_static)
    A = mass_flow_rate/(rho_* C_a)/1000
    print("rho_",n, " = ", rho_ * 1000)
    print("A_",n, " = ", A)
    return A

def calc_height(A, r_m, n):
    """
    This calculated geometric values related to the cross section of the turbine
    ____________
    Input: area "A", station number "n"
    ____________
    Output: A plot showing stability in real and imaginary axis.
    """
    h = (revs * A)/(U)
    rtrm = (r_m + 0.5*h)/(r_m - 0.5*h)
    print("h_",n,"=", h)
    print("rtrm_",n,"=", rtrm)
    return h

def calc_freevortex_nozzle(r_m, r_r, r_t, alpha):
    tan_alpha_r_2 = (r_m / r_r) * math.tan(math.radians(alpha))
    tan_alpha_t_2 = (r_m / r_t) * math.tan(math.radians(alpha))

    alpha_2_r_fv = np.arctan(tan_alpha_r_2)
    alpha_2_t_fv = np.arctan(tan_alpha_t_2)
    print("nozzle alpha values: root, tip")
    print(math.degrees(alpha_2_r_fv), math.degrees(alpha_2_t_fv))

    beta_2_r_fv = math.degrees(np.arctan(tan_alpha_r_2 - ((r_m / r_r) * (1/phi))))
    beta_2_t_fv = math.degrees(np.arctan(tan_alpha_t_2 - ((r_m / r_t) * (1/phi))))
    print("nozzle beta values: root, tip")
    print(beta_2_r_fv, beta_2_t_fv)

    return math.degrees(alpha_2_r_fv), beta_2_r_fv

def calc_freevortex_rotor(r_m, r_r, r_t, alpha):
    tan_alpha_r_3 = (r_m / r_r) * math.tan(math.radians(alpha))
    tan_alpha_t_3 = (r_m / r_t) * math.tan(math.radians(alpha))

    alpha_3_r_fv = math.degrees(np.arctan(tan_alpha_r_3))
    alpha_3_t_fv = math.degrees(np.arctan(tan_alpha_t_3))
    print("rotor alpha values: root, tip")
    print(alpha_3_r_fv, alpha_3_t_fv)

    beta_3_r_fv = math.degrees(np.arctan(tan_alpha_r_3 + ((r_m / r_r) * (1/phi))))
    beta_3_t_fv = math.degrees(np.arctan(tan_alpha_t_3 + ((r_m / r_t) * (1/phi))))
    print("rotor beta values: root, tip")
    print(abs(beta_3_r_fv), abs(beta_3_t_fv))

    return alpha_3_r_fv, beta_3_r_fv
```

Cycle Calculations

Inlet

In [ ]:
```
# ================================================================================
# CYCLE CALCULATIONS
# ================================================================================
# Conditions at the Inlet
P_a = P_0 = 101.325        # [kPa]
T_a = T_0 = 296.483        # [K]

T_01 = T_0
P_01 = 0.99*P_0
print(P_01)
```

```
100.31175
```

Low Pressure Compressor

In [ ]:
```
# LPC Calculations

P_02 = lpc_pr * P_01        # Using provided pressure ratio
T_02 = T_01 + (T_01/lpc_eff * (lpc_pr**(0.285714) - 1))
```

```
W_lpc = m_dot * c_p_air * (T_02 - T_01)         # [kJ/kg]
print(P_02, T_02, W_lpc/m_dot)
```

```
401.247 463.059728696695 167.40961234017846
```

High Pressure Compressor

In [ ]:
```
# HPC Calculations

P_03 = hpc_pr * P_02
T_03 = T_02 + (T_02/hpc_eff * (hpc_pr**(0.285714) - 1))
W_hpc = m_dot * c_p_air * (T_03 - T_02)         # [kJ/kg]
print(P_03, T_03, W_hpc/m_dot)
```

```
1203.741 662.7644873827279 200.70328247946304
```

Combustion Chamber

In [ ]:
```
# Combustion Calculations

m_air = 0.91 * m_dot       # Mass flow into combustor/turbine | See next line:
# Turbine cooling air percentage can be considered as percent
# flow of turbine inlet flow.
P_04 = P_03 * 0.982
print(P_04)
T_04 = ((c_p_air * T_03) + (AFR * FHV * combustor_eff))/((1 + AFR) * c_p_gas)
print(T_04)
```

```
1182.073662
1245.3208220773054
```

High Pressure Turbine

In [ ]:
```
# HPT Calculations

m_turbine = m_air + (0.02 * m_air)      # Includes the mass of the fuel as well
m_cool_vane_hpt = 0.03 * m_turbine          # HPT Vane Cooling Air MFR
m_cool_disc_hpt = 0.0165 * m_turbine        # HPT Disk Cooling Air MFR

# Calculation of cooling after stator
T_hpt_after_vane = ((m_turbine * c_p_gas * T_04) + (m_cool_vane_hpt * c_p_air * T_03)) / (c_p_gas * (m_turbine + m_cool_vane_h
print(T_hpt_after_vane)
# Calculation after rotor but before disc cooling
T_hpt_required_energy = T_04 - ((1.01 * W_hpc) / ((m_turbine + m_cool_vane_hpt) * c_p_gas))

# Calculation after disc cooling
T_hpt_after_rotor = (((m_turbine + m_cool_vane_hpt) * c_p_gas * T_hpt_required_energy) + (m_cool_disc_hpt * c_p_air * T_03)) /
T_05 = T_hpt_after_rotor
print(T_05)
# Pressure
P_05 = calc_turbine_pressure(P_04, T_04, T_05, 0.84)
```

```
1225.9485919279928
1053.0511295978204
524.5947533586902
```

Low Pressure Turbine

In [ ]:
```
# LPT Calculations

m_turbine_lpt = m_turbine + m_cool_vane_hpt + m_cool_disc_hpt
m_cool_disc_lpt = 0.011 * m_turbine

# Calculation of work done
T_lpt_required_energy = T_05 - ((1.01 * W_lpc) / (m_turbine_lpt * c_p_gas))
print(T_lpt_required_energy)
# Calculation after disc cooling
T_lpt_after_rotor = ((m_turbine_lpt * c_p_gas * T_lpt_required_energy) + (m_cool_disc_lpt * c_p_air * T_03)) / (c_p_gas * (m_t
T_06 = T_lpt_after_rotor
print(T_06)
# Pressure
P_06 = calc_turbine_pressure(P_05, T_05, T_06, lpc_eff)
```

```
901.4232437286498
898.0819933981012
248.8071191334912
```

Exhaust

In [ ]:
```
# Exhaust Calculations
P_08 = P_0 * ((1 + ((gamma_g-1)/(2)) * exhaust_mach**2 )**(gamma_g/(gamma_g-1)))
print(P_08)
P_07 = 1.02 * P_08      # Power Turbine Exit Total Pressure
print(P_07)
```

```
102.85344186914772
104.91051070653067
```

Power Pressure Turbine

In [ ]:
```
# Power Turbine Calculations
m_turbine_pt = m_turbine_lpt + m_cool_disc_lpt
m_cool_disc_pt = 0.0125 * m_turbine
# Calculation of work done
P_06_PT = (1 - ITD_loss) * P_06
pt_pr = P_06_PT / P_07
T_pt_required_energy = T_06 - (pt_eff * T_06 * (1 - (1 / pt_pr)**((gamma_g - 1)/gamma_g)))

# PT Temperature after disc cooling
T_pt_after_rotor = ((m_turbine_pt * c_p_gas * T_pt_required_energy) + (m_cool_disc_pt * c_p_air * T_03)) / (c_p_gas * (m_turbi
T_07 = T_pt_after_rotor         # Total temperature at power turbine exit

print(P_06_PT, T_05, T_06, T_pt_required_energy,T_07)
```

```
247.31427641869024 1053.0511295978204 898.0819933981012 738.6486734668399 736.7977261944576
```

Work and SFC Calculations

In [ ]:
```
# Calculation of work
W_pt = c_p_gas * (T_06 - T_pt_required_energy) * 0.99
SFC = (3600 * AFR) / W_pt
print(W_pt, SFC)
```

```
181.19915676827705 0.397352842497362
```